```cpp
#include "ANString.h"

int ANString::currentCount = 0;
int ANString::createdCount = 0;

ANString::ANString() {
    cap = 20;
    end = 0;
    str = new char[cap];
    str[end] = '\0';
    currentCount++;
    createdCount++;
}
ANString::ANString(const ANString& mstr) // Copy Constructor
{

        cap = mstr.cap;
        end = mstr.end;
        str = new char[cap];

        // Copy data over
        for (int i = 0; i < this->end; i++)
        {
            this->str[i] = mstr.str[i];
        }
        str[end] = '\0';
        currentCount++;
        createdCount++;
}
ANString::ANString(const char* cstr) {
    for (end = 0; cstr[end] != '\0'; ++end);
    cap = 20;
    if (end >= cap)
    {
        cap = 20 * (end / 20 + 1);
    }
    str = new char[cap];

    for (int i = 0; i <= end; ++i) {
        str[i] = cstr[i];
    }
    currentCount++;
    createdCount++;
}
ANString::~ANString() {

    delete[] this->str;
    currentCount--;
}
```

```

int ANString::length() {
    return end;
}

int ANString::capacity() {
    return cap;
}

char ANString::operator[](int index) {
    if (index >= 0 && index < end) {
        return str[index];
    }
    else {
        return '\0';
    }
}



ostream& operator<<(ostream& ostrm, const ANString& str) {
    ostrm << str.c_str();
    return ostrm;
}

bool ANString::operator<(const ANString& argStr) {
    // TODO: you need to write.

    int shortestCap = argStr.cap;

    if (cap < argStr.cap)
    {
        shortestCap = cap;
    }

    for (int i = 0; i < shortestCap; i++)
    {
        if (str[i] > argStr.str[i])
        {
            return false;
        }
        else if (str[i] < argStr.str[i])
        {
            return true;

        }

    }

```

```cpp
}


bool ANString::operator>(const ANString& argStr) {
    //TODO: you need to write.
    int shortestCap = argStr.end;

    if (end < argStr.end)
    {
        shortestCap = end;
    }

    for (int i = 0; i < shortestCap; i++)
    {
        if (str[i] < argStr.str[i])
        {
            return false;
        }
        else if (str[i] > argStr.str[i])
        {
            return true;

        }

    }

    // IF WE GET HERE, ITS THE SAME SO FAR.
    if (this->end > argStr.end)
        return true;
    return false;
}


bool ANString::operator==(const ANString& argStr) {


    int shortestCap = argStr.cap;

    if (cap < argStr.cap)
    {
        shortestCap = cap;
    }

    for (int i = 0; i < shortestCap; i++)
    {
        if (str[i] != argStr.str[i])
        {
            return false;
        }
        else if (str[i] == argStr.str[i])
        {
            return true;
        }

    }

}


ANString ANString::operator=(const ANString& rvalue) {
    end = rvalue.end;

    if (end >= cap)
    {
        cap = 20 * (end / 20 + 1);
        delete[] str;
        str = new char[cap];
    }

    for (int i = 0; i <= end; ++i) {
        str[i] = rvalue.str[i];
    }
    return (*this);
}
ANString ANString::operator+(const ANString& rvalue) {
    ANString Sum;
    Sum.end = (end + rvalue.end);

    if (Sum.end >= Sum.cap)
    {
        Sum.cap = 20 * (Sum.end / 20 + 1);
        delete[] Sum.str;
        Sum.str = new char[Sum.cap];
    }
    int i = 0;
    for (; i < end; i++)
    {
        Sum.str[i] = str[i];
    }
    for (int j = 0; j <= rvalue.end; j++)
    {
        Sum.str[i] = rvalue.str[j];
        i++;
    }


    return Sum;
}
```

```

 int ANString::getCurrentCount() {
    return currentCount;
}

 int ANString::getCreatedCount() {
    return createdCount;
}

const char* ANString::c_str() const {
    return str;
}

```